Security Assessment Report

LLM MultiChat v1.0.0

Anwendung:	LLM MultiChat
Version:	1.0.0
Analyst:	IT-Security / acadon AG
Datum:	18.12.2025
Klassifizierung:	Intern

1. Executive Summary

Die vorliegende Sicherheitsanalyse untersucht die Electron-basierte Anwendung "LLM MultiChat", welche als Multi-Interface für verschiedene LLM-Dienste (Claude, ChatGPT, Gemini, Copilot, etc.) konzipiert wurde. Die Anwendung ermöglicht das parallele Abfragen mehrerer KI-Dienste über eingebettete WebViews.

Gesamtbewertung: Die Anwendung weist ein **moderates Risikoprofil** auf. Die grundlegenden Electron-Sicherheitsmechanismen (nodeIntegration: false, contextIsolation: true) sind korrekt implementiert. Jedoch existieren architekturbedingte Schwachstellen durch die notwendige Deaktivierung von Sicherheitsheadern (X-Frame-Options, CSP) sowie potenzielle Risiken im Update-Mechanismus.

Kategorie	Risiko	Bewertung
Electron-Konfiguration	Niedrig	✓ Best Practices umgesetzt
WebView-Sicherheit	Mittel	allowpopups aktiviert
Header-Manipulation	Mittel	Architekturbedingt notwendig
Update-Mechanismus	Mittel-Hoch	Keine Integritätsprüfung
IPC-Schnittstellen	Niedrig	Minimale Exposition
Skript-Injection	Niedrig	Kontrollierter Kontext
VBS/HTA-Komponenten	Niedrig	Lokale Ausführung

2. Komponentenanalyse

2.1 Technologie-Stack

Die Anwendung basiert auf folgenden Kernkomponenten:

Komponente	Version	Funktion	Sicherheitsrelevanz
Electron	^33.0.0	Desktop-Framework	Kritisch
Chromium	~130.x (via Electron)	Rendering Engine	Kritisch
Node.js	~20.x (via Electron)	Backend Runtime	Hoch
VBScript	Windows Built-in	Installer/Updater	Mittel
HTA (mshta)	Windows Built-in	Status-Dialog	Niedrig

2.2 Dateistruktur

Die Anwendung besteht aus folgenden sicherheitsrelevanten Dateien:

• **main.js** (3.2 KB) - Electron Main Process mit BrowserWindow-Konfiguration

• **preload.js** (455 B) - Context Bridge API-Exposition

• **renderer.js** (47.7 KB) - Frontend-Logik inkl. WebView-Injection

• **Start.vbs** (7.5 KB) - Installer und Launcher-Skript

• **update.vbs** (9.0 KB) - Auto-Update von GitHub/Netzlaufwerk

• **status.hta** (3.2 KB) - GUI-Statusanzeige während Installation

• **config.json.template** (5.8 KB) - Service-Konfiguration

3. Electron-Sicherheitsanalyse

3.1 BrowserWindow-Konfiguration

Die BrowserWindow-Konfiguration in main.js implementiert die empfohlenen Sicherheitseinstellungen:

```
webPreferences: { nodeIntegration: false, // ✓ Node.js APIs im Renderer deaktiviert
contextIsolation: true, // ✓ Preload-Skript isoliert preload: path.join(__dirname,
'preload.js'), webviewTag: true // ■ WebView-Tag aktiviert (erforderlich) }
```

Positiv:

✓ nodeIntegration: false verhindert direkten Zugriff auf Node.js APIs aus dem Renderer

✓ contextIsolation: true schützt vor Prototype Pollution und Context Tampering

✓ Preload-Skript exponiert nur explizit definierte APIs via contextBridge

3.2 Header-Manipulation

Die Anwendung entfernt aktiv Sicherheitsheader von HTTP-Responses, um das Embedding externer LLM-Interfaces zu ermöglichen:

```
const headersToRemove = [ 'x-frame-options', 'content-security-policy',
'content-security-policy-report-only' ]; headersToRemove.forEach(header => delete
responseHeaders[header]);
```

Risikobewertung: MITTEL

Diese Manipulation ist **architekturbedingt erforderlich**, da die LLM-Anbieter (Claude, ChatGPT, etc.) X-Frame-Options: DENY setzen, um Clickjacking zu verhindern. Ohne diese Deaktivierung wäre das Embedding in WebViews nicht möglich.

Realistische Risikoeinschätzung: Da die WebViews nur vordefinierte, vertrauenswürdige URLs (claude.ai, chat.openai.com, etc.) laden und keine willkürliche Navigation erlauben, ist das tatsächliche Angriffsrisiko begrenzt. Ein Angreifer müsste zunächst die Konfiguration manipulieren oder eine Man-in-the-Middle-Position erlangen.

3.3 WebView-Sicherheit

Die WebViews werden mit folgenden Attributen erstellt:

Attribut	Wert	Sicherheitsbewertung
nodeIntegration	nicht gesetzt (default: false)	✓ Sicher
contextIsolation	nicht gesetzt (default: true)	✓ Sicher
partition	persist:{service.id}	✓ Session-Isolation
allowpopups	aktiviert	■ Popup-Fenster erlaubt

Session-Isolation: Die Verwendung von *partition="persist:{service.id}"* sorgt dafür, dass jeder LLM-Dienst eine eigene persistente Session erhält. Dies verhindert Cookie-/Session-Leakage zwischen den verschiedenen Diensten.

3.4 IPC-Schnittstellen (Inter-Process Communication)

Die exponierte API über contextBridge ist minimal gehalten:

```
contextBridge.exposeInMainWorld('electronAPI', { getConfig: () =>
ipcRenderer.invoke('get-config'), getUserSettings: () =>
ipcRenderer.invoke('get-user-settings'), saveUserSettings: (settings) =>
ipcRenderer.invoke('save-user-settings', settings), copyToClipboard: (text) =>
ipcRenderer.invoke('copy-to-clipboard', text), readClipboard: () =>
ipcRenderer.invoke('read-clipboard') });
```

Risikobewertung: NIEDRIG

Die IPC-Schnittstellen sind sicherheitstechnisch unbedenklich:

• **get-config / get-user-settings:** Read-only Zugriff auf lokale JSON-Dateien

• **save-user-settings:** Schreibt nur in user-settings.json (keine Path-Traversal möglich)

• **copy-to-clipboard / read-clipboard:** Standardfunktionalität, kein Sicherheitsrisiko

4. Skript-Injection-Analyse

Die Anwendung injiziert JavaScript in die geladenen WebViews, um Texteingaben in die verschiedenen LLM-Interfaces zu automatisieren:

```
const escapedText = text.replace(/\\/g, '\\\\') .replace(/`/g, '\\`') .replace(/\$/g, '\\$');
const result = await webviews[service.id].executeJavaScript(script);
```

Risikobewertung: NIEDRIG

Analyse: Die Skript-Injection erfolgt ausschließlich mit vom Benutzer eingegebenem Text. Eine Escape-Sequenz verhindert Template-Literal-Injection. Das injizierte Skript interagiert nur mit DOM-Elementen der LLM-Oberflächen (Input-Felder, Submit-Buttons).

Theoretisches Risiko: Ein bösartiger LLM-Dienst könnte versuchen, über die Response-Extraktion Code einzuschleusen. Da jedoch keine eval()-Aufrufe oder innerHTML-Zuweisungen mit den Responses erfolgen, ist dieses Risiko nicht exploitierbar.

5. Update-Mechanismus

Der Update-Mechanismus in update.vbs lädt Dateien von GitHub oder einem Netzlaufwerk:

```
GITHUB_RAW = "https://raw.githubusercontent.com/3Dcut/multiLLM/main" FALLBACK_SOURCE =
"\\acv27.acadon.acadon.de\Ablage\jmi\llm-multichat" ' Downloads ohne Integritätsprüfung:
DownloadFile sourceUrl, destPath
```

Risikobewertung: MITTEL-HOCH

■ **Keine Signaturprüfung:** Downloads werden nicht kryptografisch verifiziert

■ **Keine Hash-Validierung:** SHA256/MD5 Checksums fehlen

■ **HTTPS ohne Certificate Pinning:** MITM-Angriffe theoretisch möglich

■ **Automatische Code-Ersetzung:** Heruntergeladene Dateien überschreiben lokale Versionen

Realistische Einschätzung: Für einen erfolgreichen Angriff müsste ein Angreifer:

1. Zugriff auf das GitHub-Repository erlangen (Kompromittierung der Credentials) ODER

2. Eine Man-in-the-Middle-Position im Netzwerk einnehmen ODER

3. Zugriff auf das interne Netzlaufwerk (\\acv27...) erhalten

Im Kontext eines internen Unternehmenstools mit kontrollierten Deployment-Quellen ist das Risiko akzeptabel, sollte aber bei breiterer Verteilung adressiert werden.

6. Windows-Skript-Komponenten (VBS/HTA)

Die Anwendung nutzt VBScript (Start.vbs, update.vbs) und eine HTA-Datei (status.hta) für Installation, Updates und Statusanzeige.

Risikobewertung: NIEDRIG

Analyse: Die VBS-Skripte führen ausschließlich kontrollierte Operationen aus:

• Download von Node.js von der offiziellen Domain (nodejs.org)

• Ausführung von npm install im Anwendungsverzeichnis

• Datei-Downloads von definierten Quellen (GitHub, internes Netzlaufwerk)

• Lesen/Schreiben von Statusdateien im %TEMP%-Verzeichnis

HTA-Datei: Die status.hta verwendet ActiveX-Objekte (FileSystemObject, WScript.Shell), jedoch nur zum Lesen einer lokalen Statusdatei. Keine Netzwerkkommunikation oder Codeausführung aus externen Quellen.

7. Bekannte Electron-Schwachstellen

Electron 33.x basiert auf Chromium ~130.x. Relevante CVEs für diese Version:

CVE	Beschreibung	Relevanz für LLM MultiChat
CVE-2025-55305	ASAR Integrity Bypass via V8 Snapshot	Niedrig - Erfordert lokalen Zugriff
CVE-2024-12695	Out-of-bounds Write in V8	Mittel - Chromium-Update empfohlen
runAsNode CVEs	Living-off-the-land via Fuses	Niedrig - Erfordert vorherige Kompromittierung

Hinweis: Electron veröffentlicht regelmäßig Security-Updates. Es wird empfohlen, die Electron-Version aktuell zu halten. Die Spezifikation "^33.0.0" in package.json erlaubt automatische Minor-Updates, was positiv zu bewerten ist.

8. Angriffsvektoren und realistische Risiken

Die folgende Analyse bewertet potenzielle Angriffsvektoren unter Berücksichtigung realistischer Szenarien und des tatsächlichen Deploymentkontexts:

8.1 Supply-Chain-Angriff via Update-Mechanismus

Risiko: MITTEL

Szenario: Angreifer kompromittiert das GitHub-Repository oder führt MITM-Angriff durch.

Voraussetzungen: Zugriff auf GitHub-Credentials des Repository-Besitzers ODER Netzwerk-Position für MITM ODER Zugriff auf internes Netzlaufwerk.

Realistische Einschätzung: Bei einem internen Tool mit kontrollierten Quellen ist dieses Risiko moderat. GitHub selbst bietet 2FA und andere Schutzmechanismen.

8.2 XSS-zu-RCE Eskalation

Risiko: NIEDRIG

Szenario: XSS-Schwachstelle in einem LLM-Interface führt zu Code-Execution.

Warum unwahrscheinlich: nodeIntegration ist deaktiviert und contextIsolation aktiv. Selbst bei XSS im WebView hat der Angreifer keinen Zugriff auf Node.js-APIs. Die exponierte electronAPI enthält keine gefährlichen Funktionen.

8.3 Session-Hijacking zwischen Services

Risiko: NIEDRIG

Szenario: Ein kompromittierter LLM-Dienst greift auf Sessions anderer Dienste zu.

Warum unwahrscheinlich: Jeder Service nutzt eine separate Session-Partition (partition="persist:{service.id}"). Cookies und LocalStorage sind vollständig isoliert.

8.4 Credential-Extraktion via Clipboard

Risiko: NIEDRIG

Szenario: Angreifer nutzt readClipboard() um sensible Daten auszulesen.

Warum unwahrscheinlich: Die Clipboard-API ist nur aus dem Renderer-Kontext der Hauptanwendung (index.html) aufrufbar, nicht aus den eingebetteten WebViews. Der Benutzer muss aktiv mit der Anwendung interagieren.

9. Empfehlungen

9.1 Kritische Empfehlungen (Priorität: Hoch)

1. **Code-Signierung für Updates:** Implementierung einer SHA256-Hash-Validierung für heruntergeladene Dateien. Speicherung der erwarteten Hashes in einer signierten Manifest-Datei.

2. **Certificate Pinning:** Für die GitHub-API und Raw-Downloads sollte Certificate Pinning implementiert werden, um MITM-Angriffe zu erschweren.

3. **Electron-Updates:** Regelmäßige Updates auf die neueste Electron-Version, mindestens monatliche Überprüfung auf Security-Releases.

9.2 Wichtige Empfehlungen (Priorität: Mittel)

1. **WebView Navigation einschränken:** Implementierung eines will-navigate Event-Handlers, der nur Navigation zu den konfigurierten Service-URLs erlaubt.

2. **allowpopups entfernen:** Falls möglich, das allowpopups-Attribut von den WebViews entfernen, um Popup-basierte Angriffe zu verhindern.

3. **Logging verbessern:** Implementierung eines Security-Audit-Logs für Update-Vorgänge und fehlgeschlagene Netzwerkzugriffe.

4. **Electron Fuses:** Deaktivierung von runAsNode und enableNodeCliInspectArguments via Electron Fuses für produktive Builds.

9.3 Optionale Verbesserungen (Priorität: Niedrig)

1. **CSP für lokale Inhalte:** Definition einer Content-Security-Policy für die Hauptanwendung (index.html).

2. **Subresource Integrity:** Falls externe Ressourcen geladen werden, SRI-Hashes verwenden.

3. **Automatische Update-Benachrichtigungen:** Statt automatischer Updates nur Benachrichtigungen mit manueller Bestätigung.

10. Fazit

Die Anwendung "LLM MultiChat" weist ein **akzeptables Sicherheitsniveau** für den internen Unternehmenseinsatz auf. Die Electron-Sicherheitsmechanismen sind korrekt implementiert, und die architekturbedingten Kompromisse (Header-Deaktivierung) sind gut dokumentiert und im Kontext des Use-Cases vertretbar.

Hauptrisiken: Der Update-Mechanismus stellt das größte Risiko dar, da keine Integritätsprüfung der heruntergeladenen Dateien erfolgt. Dieses Risiko wird jedoch durch die kontrollierten Deployment-Quellen (GitHub mit 2FA, internes Netzlaufwerk) mitigiert.

Keine kritischen Schwachstellen: Es wurden keine Schwachstellen identifiziert, die eine unmittelbare Remote Code Execution ohne vorherige Kompromittierung des Systems oder der Infrastruktur ermöglichen würden.

Kategorie	Status	Kommentar
Gesamtrisiko	MITTEL	Akzeptabel für internen Einsatz
Sofortiger Handlungsbedarf	NEIN	Keine kritischen Lücken
Empfohlene Maßnahmen	JA	Update-Signierung implementieren
Freigabe für Produktiveinsatz	JA	Mit dokumentierten Einschränkungen

---

Dieser Report basiert auf einer statischen Code-Analyse und öffentlich verfügbaren Informationen zu Electron-Sicherheit. Eine dynamische Analyse (Penetration Test) wurde nicht durchgeführt. Die Bewertungen spiegeln den Stand zum Analysezeitpunkt wider.